上海建桥学院
SHANGHAI JIAN QIAO UNIVERSITY

Shared Preferences

主讲：万永权

学习目标

AIMS

01 掌握SharedPreferences的使用方法

02 掌握各种文件存储的区别与适用情况

8.1 简单存储

8.1.1 SharedPreferences

- SharedPreferences是一种轻量级的数据保存方式
- 通过SharedPreferences开发人员可以将NVP（Name/Value Pair，名称/值对）保存在Android的文件系统中，而且SharedPreferences完全屏蔽了对文件系统的操作过程
- 开发人员仅通过调用SharedPreferences中的函数就可以实现对NVP的保存和读取

8.1 简单存储

8.1.1 SharedPreferences

- SharedPreferences不仅能够保存数据，还能够实现不同应用程序间的数据共享
- SharedPreferences支持三种访问模式
  - 私有（MODE_PRIVATE）：仅创建SharedPreferences的程序有权限对其进行读取或写入
  - 全局读（MODE_WORLD_READABLE）：不仅创建程序可以对其进行读取或写入，其它应用程序也具有读取操作的权限，但没有写入操作的权限
  - 全局写（MODE_WORLD_WRITEABLE）：所有程序都可以对其进行写入操作，但没有读取操作的权限

8.1 简单存储

8.1.1 SharedPreferences

- 首先， 定义SharedPreferences的访问模式，下面的代码将访问模式定义为私有模式
  - public static int MODE = MODE_PRIVATE;
- 有的时候需要将SharedPreferences的访问模式设定为即可以全局读，也可以全局写，这就需要将两种模式写成下面的方式
- public static int MODE = Context. MODE_PRIVATE + Context. MODE_PRIVATE;

8.1 简单存储

8.1.1 SharedPreferences

- 除了定义SharedPreferences的访问模式，还要定义SharedPreferences的名称，这个名称也是SharedPreferences在Android文件系统中保存的文件名称
- 一般将SharedPreferences名称声明为字符串常量，这样可以在代码中多次使用

```
1    public static final String PREFERENCE_NAME = "SaveSetting";
```

- 使用SharedPreferences时需要将访问模式和SharedPreferences名称作为参数传递到getSharedPreferences()函数，则可获取到SharedPreferences实例

```
1    SharedPreferences sharedPreferences =
getSharedPreferences(PREFERENCE_NAME, MODE);
```

8.1 简单存储

8.1.1 SharedPreferences

- 在获取到SharedPreferences实例后，可以通过SharedPreferences.Editor类对SharedPreferences进行修改，最后调用commit()函数保存修改内容
- SharedPreferences广泛支持各种基本数据类型，包括整型、布尔型、浮点型和长型等

```
1  SharedPreferences.Editor editor = sharedPreferences.edit();
2  editor.putString("Name", "Tom");
3  editor.putInt("Age", 20);
4  editor.putFloat("Height", 1.81f);
5  editor.commit();
```

8.1 简单存储

8.1.1 SharedPreferences

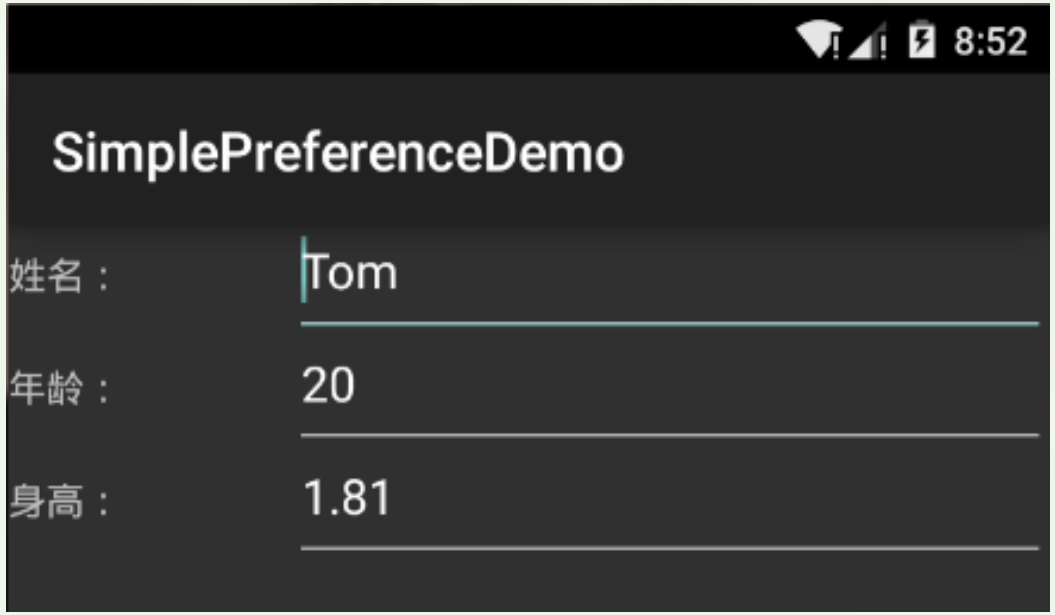
- 如果需要从已经保存的SharedPreferences中读取数据，同样是调用getSharedPreferences()函数，并在函数第1个参数中指明需要访问的SharedPreferences名称，最后通过get<Type>()函数获取保存在SharedPreferences中的NVP
- get<Type>()函数的第1个参数是NVP的名称
- 第2个参数是在无法获取到数值的时候使用的缺省值

```
1  SharedPreferences sharedPreferences =
     getSharedPreferences(PREFERENCE_NAME, MODE);
2  String name = sharedPreferences.getString("Name","Default Name");
3  int age = sharedPreferences.getInt("Age", 20);
4  float height = sharedPreferences.getFloat("Height",1.81f);
```

8.1 简单存储

8.1.2 示例

- 下面将通过SimplePreferenceDemo示例介绍SharedPreferences的文件保存位置和保存格式
- 下图是SimplePreferenceDemo示例的用户界面



8.1 简单存储

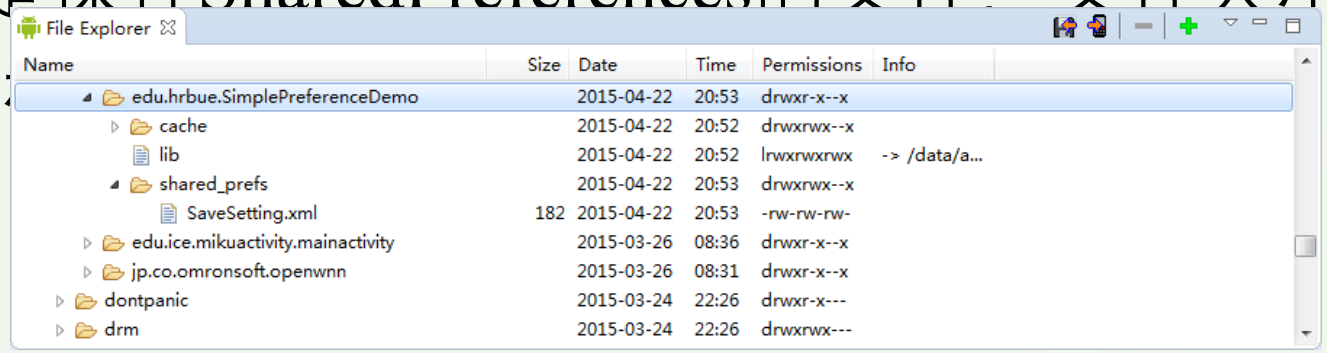
8.1.2 示例

- 用户在界面上的输入信息，在Activity关闭时通过SharedPreferences进行保存。当应用程序重新开启时，再通过SharedPreferences将信息读取出来，并重新呈现在用户界面上
- SimplePreferenceDemo示例运行并通过“回退键”退出，通过FileExplorer查看/data/data下的数据，Android系统为每个应用程序建立了与包同名的目录，用来保存应用程序产生的数据文件，包括普通文件、SharedPreferences文件和数据库文件等
- SharedPreferences产生的文件就保存在/data/data/<package name>/shared_prefs目录下

8.1 简单存储

8.1.2 示例

- 在本示例中，shared_prefs目录中生成了一个名为SaveSetting.xml的文件
- 如图8.2所示，保存在/data/data/edu.hrbeu.SimplePreferenceDemo/shared_prefs目录下
- 这个文件就是保存SharedPreferences的文件，文件大小为170字节，在Linux下的



8.1 简单存储

8.1.2 示例

- 在Linux系统中，文件权限分别描述了创建者、同组用户和其它用户对文件的操作限制。x表示可执行，r表示可读，w表示可写，d表示目录，-表示普通文件
- 因此，“-rw-rw-rw”表示SaveSetting.xml可以被创建者、同组用户和其它用户进行读取和写入操作，但不可执行
- 产生这样的文件权限与程序人员设定的SharedPreferences的访问模式有关，“-rw-rw-rw”的权限是“全局读+全局写”的结果
- 如果将SharedPreferences的访问模式设置为私有，则文件权限将成为“-rw-rw ---”，表示仅有创建者和同组用户具有读写文件的权限

8.1 简单存储

8.1.2 示例

- SaveSetting.xml文件是以XML格式保存的信息，内容如下：

```
1  <?xml version='1.0' encoding='utf-8' standalone='yes' ?>
2  <map>
3          <float name="Height" value="1.81" />
4          <string name="Name">Tom</string>
5          <int name="Age" value="20" />
6  </map>
```

8.1 简单存储

8.1.2 示例

- SimplePreferenceDemo示例在onStart()函数中调用loadSharedPreferences()函数，读取保存在SharedPreferences中的姓名、年龄和身高信息，并显示在用户界面上
- 当Activity关闭时，在onStop()函数调用saveSharedPreferences()，保存界面上的信息
- SimplePreferenceDemoActivity.java的完整代码如下：

```
1 package edu.lrbeu.SimplePreferenceDemo;
2
3   import android.app.Activity;
4   import android.content.Context;
5   import android.content.SharedPreferences;
6   import android.os.Bundle;
```

8.1 简单存储

8.1.2 示例

```
7   import android.widget.EditText;
8
9   public class SimplePreferenceDemoActivity extends Activity {
10
11          private EditText nameText;
12          private EditText ageText;
13          private EditText heightText;
14          public static final String PREFERENCE_NAME = "SaveSetting";
15          public static int MODE = Context.MODE_WORLD_READABLE +
        Context.MODE_WORLD_WRITEABLE;
16
17          @Override
18          public void onCreate(Bundle savedInstanceState) {
19            super.onCreate(savedInstanceState);
20            setContentView(R.layout.main);
21            nameText = (EditText)findViewById(R.id.name);
```

8.1 简单存储

8.1.2 示例

```
22          ageText = (EditText)findViewById(R.id.age);
23          heightText = (EditText)findViewById(R.id.height);
24           }
25
26           @Override
27           public void onStart(){
28          super.onStart();
29          loadSharedPreferences();
30           }
31           @Override
32           public void onStop(){
33          super.onStop();
34          saveSharedPreferences();
35           }
36
```

8.1 简单存储

8.1.2 示例

```
37          private void loadSharedPreferences(){
38          SharedPreferences sharedPreferences = getSharedPreferences(PREFERENCE_NAME,
    MODE);
39          String name = sharedPreferences.getString("Name","Tom");
40          int age = sharedPreferences.getInt("Age", 20);
41          float height = sharedPreferences.getFloat("Height",1.81f);
42
43          nameText.setText(name);
44          ageText.setText(String.valueOf(age));
45          heightText.setText(String.valueOf(height));
46          }
47
48          private void saveSharedPreferences(){
49          SharedPreferences sharedPreferences = getSharedPreferences(PREFERENCE_NAME,
    MODE);
50          SharedPreferences.Editor editor = sharedPreferences.edit();
```

8.1 简单存储

8.1.2 示例

```
51
52          editor.putString("Name", nameText.getText().toString());
53          editor.putInt("Age", Integer.parseInt(ageText.getText().toString()));
54          editor.putFloat("Height", Float.parseFloat(heightText.getText().toString()));
55          editor.commit();
56           }
57    }
```

8.1 文件存储

- 虽然SharedPreferences能够为开发人员简化数据存储和访问过程，但直接使用文件系统保存数据仍然是Android数据存储中不可或缺的组成部分
- Android使用Linux的文件系统，开发人员可以建立和访问程序自身建立的私有文件，也可以访问保存在资源目录中的原始文件和XML文件，还可以将文件保存在TF卡等外部存储设备中

8.2 文件存储

8.2.1 内部存储

- Android系统允许应用程序创建仅能够自身访问的私有文件，文件保存在设备的内部存储器上，在Android系统下的/data/data/<package name>/files目录中
- Android系统不仅支持标准Java的IO类和方法，还提供了能够简化读写流式文件过程的函数
- 这里主要介绍两个函数
  - openFileOutput()
  - openFileInput()

8.2 文件存储

8.2.1 内部存储

openFileOutput()函数

- openFileOutput()函数为写入数据做准备而打开文件
- 如果指定的文件存在，直接打开文件准备写入数据
- 如果指定的文件不存在，则创建一个新的文件
- openFileOutput()函数的语法格式如下：
  - public FileOutputStream openFileOutput(String name, int mode)
  - 第1个参数是文件名称，这个参数不可以包含描述路径的斜杠
  - 第2个参数是操作模式，Android系统支持四种文件操作模式
- 函数的返回值是FileOutputStream类型

8.2 文件存储

8.2.1 内部存储

- openFileOutput()函数
  - 两种文件操作模式

模式	说明
MODE_PRIVATE	私有模式，缺陷模式，文件仅能够被创建文件的程序访问，或具有相同UID的程序访问。
MODE_APPEND	追加模式，如果文件已经存在，则在文件的结尾处添加新数据。

8.7 文件存储

8.2.1 内部存储

openFileOutput()函数

- 使用openFileOutput()函数建立新文件的示例代码如下：

```
1  String FILE_NAME = "fileDemo.txt";
2  FileOutputStream fos = openFileOutput(FILE_NAME,Context.MODE_PRIVATE)
3  String text = “Some data”;
4  fos.write(text.getBytes());
5  fos.flush();
6  fos.close();
```

- 代码首先定义文件的名称为fileDemo.txt
- 然后使用openFileOutput()函数以私有模式建立文件，并调用write()函数将数据写入文件，调用flush()函数将缓冲中的数据写入文件，最后调用close()函数关闭FileOutputStream

8.2 文件存储

8.2.1 内部存储

- openFileOutput()函数
  - 为了提高文件系统的性能，一般调用write()函数时，如果写入的数据量较小，系统会把数据保存在数据缓冲区中，等数据量积攒到一定程度时再将数据一次性写入文件
  - 因此，在调用close()函数关闭文件前，务必要调用flush()函数，将缓冲区内所有的数据写入文件
  - 如果开发人员在调用close()函数前没有调用flush()，则可能导致部分数据丢失

8.2 文件存储

8.2.1 内部存储

✓ openFileInput()函数

- openFileInput()函数为读取数据做准备而打开文件
- openFileInput()函数的语法格式如下：
  - public FileInputStream openFileInput (String name)
  - 第1个参数也是文件名称，同样不允许包含描述路径的斜杠
  - 使用openFileInput()函数打开已有文件，并以二进制方式读取数据的示例代码如下：

```
1  String FILE_NAME = "fileDemo.txt";
2  FileInputStream fis = openFileInput(FILE_NAME);
3
4  byte[] readBytes = new byte[fis.available()];
5  while(fis.read(readBytes) != -1){
6  }
```

8.7 文件存储

8.2.1 内部存储

- openFileInput()函数
  - 上面的两部分代码在实际使用过程中会遇到错误提示，因为文件操作可能会遇到各种问题而最终导致操作失败，因此代码应该使用try/catch捕获可能产生的异常

8.2 文件存储

8.2.1 内部存储

- InternalFileDemo是用来演示在内部存储器上进行文件写入和读取的示例。
- 用户界面如下图所示，用户将需要写入的数据添加在EditText中，通过“写入文件”按钮将数据写入到/data/data/edu.hrbeu.InternalFileDemo/files/fileDemo.txt文件中
- 如果用户选择“追加模式”，数据将会添加到fileDemo.txt文件的结尾处
- 通过“读取文件”按钮，程序会读取fileDemo.txt文件的内容，并显示在界面下方的白色区域中

8.2 文件存储

8.2.1 内部存储

- InternalFileDemo用户界面图

8.2 文件存储

8.2.1 内部存储

- InternalFileDemo示例的核心代码:

```
1   OnClickListener writeButtonListener = new OnClickListener() {
2           @Override
3           public void onClick(View v) {
4           FileOutputStream fos = null;
5           try {
6               if (appendBox.isChecked()){
7                   fos = openFileOutput(FILE_NAME,Context.MODE_APPEND);
8               }else {
9                   fos = openFileOutput(FILE_NAME,Context.MODE_PRIVATE);
10              }
11          String text = entryText.getText().toString();
12          fos.write(text.getBytes());
```

8.2 文件存储

8.2.1 内部存储

```
13        labelView.setText("文件写入成功，写入长度："+text.length());
14        entryText.setText("");
15        } catch (FileNotFoundException e) {
16              e.printStackTrace();
17        }
18        catch (IOException e) {
19              e.printStackTrace();
20        }
21        finally{
22              if (fos != null){
23    try {
24              fos.flush();
25            fos.close();
26    } catch (IOException e) {
27            e.printStackTrace();
```

8.2 文件存储

8.2.1 内部存储

```
28  }
29          }
30      }
31          }
32  };
33   OnClickListener readButtonListener = new OnClickListener() {
34          @Override
35          public void onClick(View v) {
36      displayView.setText("");
37      FileInputStream fis = null;
38      try {
39                      fis = openFileInput(FILE_NAME);
40              if (fis.available() == 0){
41  return;
42                  }
```

8.2 文件存储

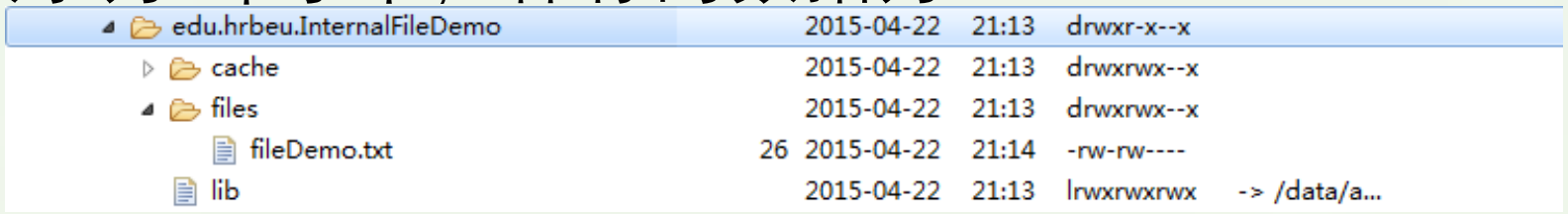
8.2.1 内部存储

```
43                  byte[] readBytes = new byte[fis.available()];
44              while(fis.read(readBytes) != -1){
45              }
46              String text = new String(readBytes);
47              displayView.setText(text);
48              labelView.setText("文件读取成功，文件长度："+text.length());
49          } catch (FileNotFoundException e) {
50              e.printStackTrace();
51          }
52          catch (IOException e) {
53              e.printStackTrace();
54          }
55           }
56      };
```

8.2 文件存储

8.2.1 内部存储

- 程序运行后，在/data/data/edu.hrbeu.InternalFileDemo/files/目录下，找到了新建立的fileDemo.txt文件，下图所示
- 从文件权限上进行分析fileDemo.txt文件，“-rw-rw---”表明文件仅允许文件创建者和同组用户读写，其它用户无权使用
- 文件的大小为9个字节，保存的数据为“Some data”



8.2 文件存储

8.2.2 外部存储

- Android的外部存储设备一般指Micro SD卡，又称T-Flash，是一种广泛使用于数码设备的超小型记忆卡
- 下图是东芝出品的32G Micro SD卡

8.2 文件存储

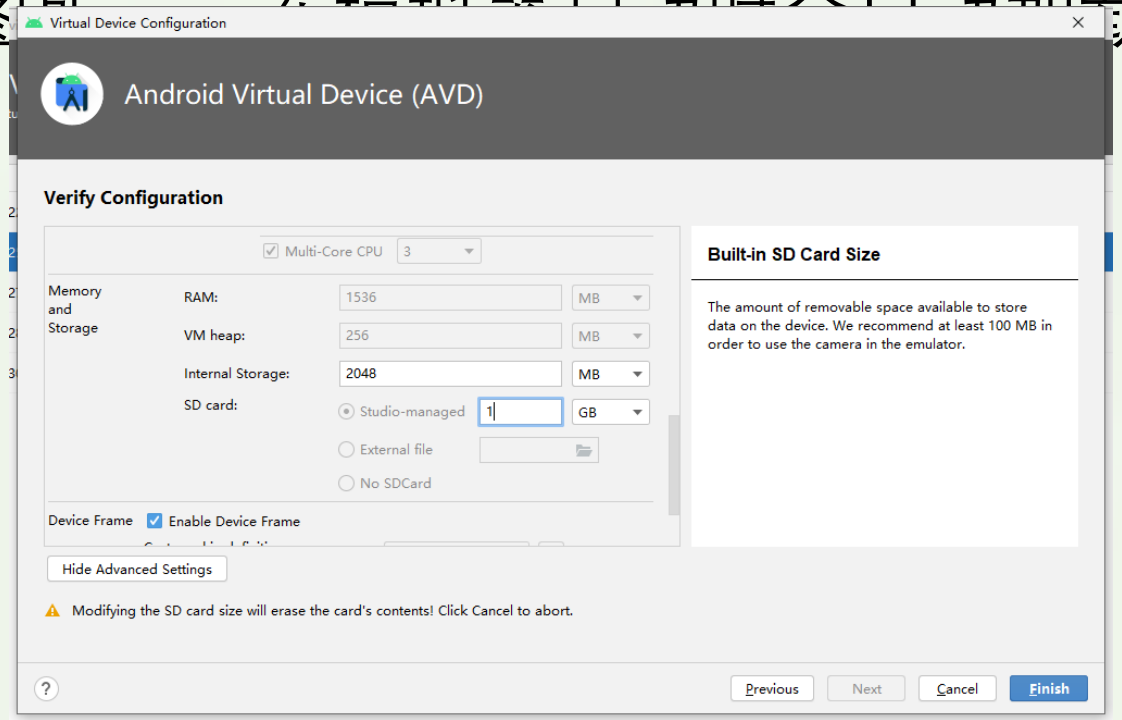
8.2.2 外部存储

- Micro SD卡适用于保存大尺寸的文件或者是一些无需设置访问权限的文件
- 如果用户希望保存录制的视频文件和音频文件，因为Android设备的内部存储空间有限，所以使用Micro SD卡则是非常适合的选择
- 但如果需要设置文件的访问权限，则不能够使用Micro SD卡，因为Micro SD卡使用FAT（File Allocation Table）文件系统，不支持访问模式和权限控制
- Android的内部存储器使用的是Linux文件系统，则可通过文件访问权限的控制保证文件的私密性

8.2 文件存储

8.2.2 外部存储

- Android模拟器支持SD卡的模拟，在模拟器建立时可以选择SD卡的容量，如下图所示，在模拟器启动时会自动加载SD卡



8.7 文件存储

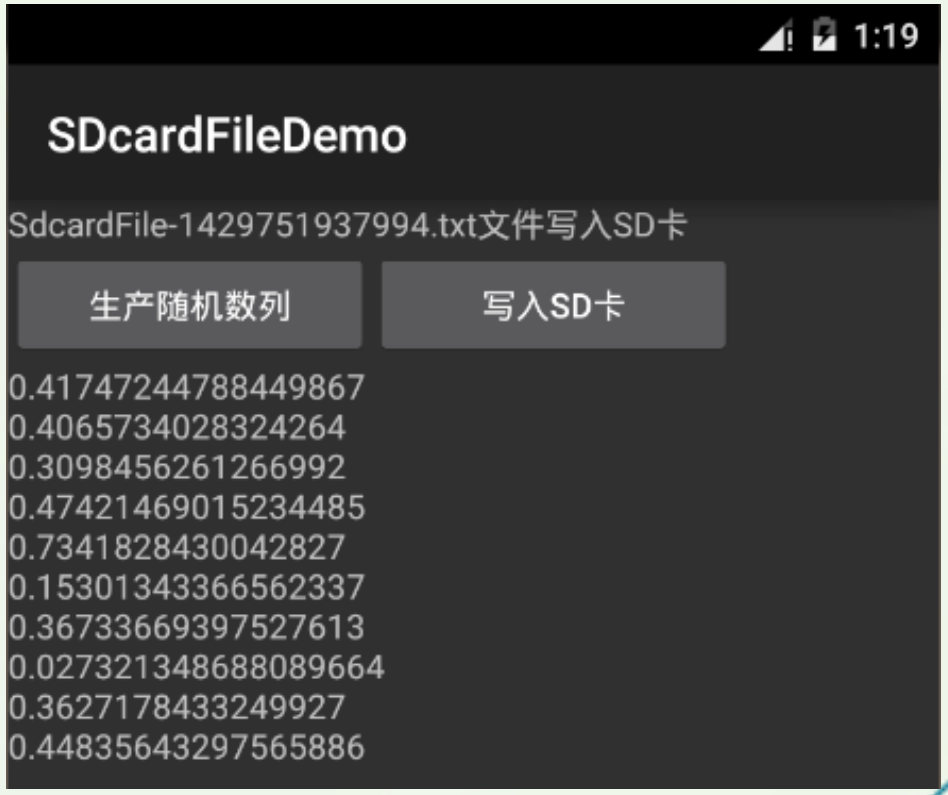
8.2.2 外部存储

- ✓正确加载SD卡后，SD卡中的目录和文件被映射到/mnt/sdcard目录下
- ✓因为用户可以加载或卸载SD卡，所以在编程访问SD卡前首先需要检测/mnt/sdcard目录是否可用
- ✓如果不可用，说明设备中的SD卡已经被卸载。如果可用，则直接通过使用标准的java.io.File类进行访问
- ✓SDcardFileDemo示例用来说明如何将数据保存在SD卡中
- ✓首先通过“生产随机数列”按钮生产10个随机小数，然后通过“写入SD卡”按钮将生产的数据保存在SD卡的根目录下，也就是Android系统的/mnt/sdcard目录下

8.2 文件存储

8.2.2 外部存储

- ✓SDcardFileDemo用户界面图



8.2 文件存储

8.2.2 外部存储

✓SDcardFileDemo示例运行后，在每次点击“写入SD卡”按钮后，都会在SD卡中生产一个新文件，文件名各不相同，如下图所示:

名称	修改日期	类型	大小
data	2021/3/26 14:31	文件夹	
hardware-qemu.ini.lock	2021/4/11 21:02	文件夹	
snapshots	2021/3/26 14:31	文件夹	
AVD.conf	2021/4/11 21:03	CONF 文件	1 KB
cache	2021/3/26 14:32	光盘映像文件	67,584 KB
cache.img.qcow2	2021/4/11 21:00	QCOW2 文件	9,984 KB
config	2021/3/26 14:31	配置设置	2 KB
emulator-user	2021/4/11 21:00	配置设置	1 KB
emu-launch-params	2021/4/11 21:02	文本文档	1 KB
hardware-qemu	2021/4/11 21:02	配置设置	4 KB
multiinstance.lock	2021/4/11 21:02	LOCK 文件	0 KB
quickbootChoice	2021/4/11 21:00	配置设置	1 KB
read-snapshot	2021/4/11 21:02	文本文档	0 KB
sdcard	2021/3/26 14:31	光盘映像文件	524,288 KB
sdcard.img.qcow2	2021/4/11 21:00	QCOW2 文件	577 KB
userdata	2021/3/26 14:26	光盘映像文件	563,200 KB
userdata-qemu	2021/3/26 14:32	光盘映像文件	819,200 KB
userdata-qemu.img.qcow2	2021/4/11 21:00	QCOW2 文件	90,624 KB
version_num.cache	2021/3/26 14:32	CACHE 文件	1 KB

8.2 文件存储

8.2.2 外部存储

- ✓SDcardFileDemo示例与InternalFileDemo示例的核心代码比较相似，不同之处在于代码中添加了/mnt/sdcard目录存在性检查（代码第7行），并使用“绝对目录+文件名”的形式表示新建立的文件（代码第8行），并在写入文件前对文件的存在性和可写入性进行检查(代码第12行)
- ✓为了保证在SD卡中多次写入时文件名不会重复，在文件名中使用了唯一且不重复的标识（代码第5行），这个标识通过调用System.currentTimeMillis()函数获得，表示从1970年00:00:00到当前所经过的毫秒数
- ✓SDcardFileDemo示例的核心代码如下:

8.7 文件存储

8.2.2 外部存储

```
1   private static String randomNumbersString = "";
2   OnClickListener writeButtonListener = new OnClickListener() {
3           @Override
4           public void onClick(View v) {
5           String fileName = "SdcardFile-"+System.currentTimeMillis()+".txt";
6           File dir = new File("/sdcard/");
7           if (dir.exists() && dir.canWrite()) {
8                   File newFile = new File(dir.getAbsolutePath() + "/" + fileName);
9                   FileOutputStream fos  = null;
10                  try {
11      newFile.createNewFile();
12      if (newFile.exists() && newFile.canWrite()) {
13              fos  = new FileOutputStream(newFile);
14               fos.write(randomNumbersString.getBytes());
15             TextView labelView = (TextView)findViewById(R.id.label);
16             labelView.setText(fileName + "文件写入SD卡");
```

8.7 文件存储

8.2.2 外部存储

```
17      }
18              } catch (IOException e) {
19      e.printStackTrace();
20              } finally {
21      if (fos != null) {
22              try{
23                  fos.flush();
24                  fos.close();
25              }
26              catch (IOException e) { }
27          }
28              }
29          }
30           }
31  };
```

8.2 文件存储

8.2.2 外部存储

- 程序在模拟器中运行前，还必须在AndroidManifest.xml中注册两个用户权限，分别是加载卸载文件系统的权限和向外部存储器写入数据的权限
- AndroidManifest.xml的核心代码如下：

```
1    <uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS">
     </uses-permission>
2    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE">
     </uses-permission>
```

8.7 文件存储

8.2.3 资源文件

- 开发人员除了可以在内部和外部存储设备上读写文件以外，还可以访问在/res/raw和/res/xml目录中的原始格式文件和XML文件，这些文件是程序开发阶段在工程中保存的文件
- 原始格式文件可以是任何格式的文件，例如视频格式文件、音频格式文件、图像文件或数据文件等等
- 在应用程序编译和打包时，/res/raw目录下的所有文件都会保留原有格式不变。而/res/xml目录下一般用来保存格式化数据的XML文件，则会在编译和打包时将XML文件转换为二进制格式，用以降低存储器空间占用和提高访问效率，在应用程序运行的时候会以特殊的方式进行访问

8.2 文件存储

8.2.3 资源文件

- ResourceFileDemo示例演示了如何在程序运行时访问资源文件
- 当用户点击“读取原始文件”按钮时，程序将读取/res/raw/raw_file.txt文件，并将内容显示在界面上，如下图所示:

8.2 文件存储

8.2.3 资源文件

- 当用户点击“读取XML文件”按钮时，程序将读取/res/xml/people.xml文件，也将内容显示在界面上，如下图所示:

8.7 文件存储

8.2.3 资源文件

- 读取原始格式文件首先需要调用getResource()函数获得资源实例，然后通过调用资源实例的openRawResource()函数，以二进制流的形式打开指定的原始格式文件。在读取文件结束后，调用close()函数关闭文件流
- ResourceFileDemo示例中读取原始格式文件的核心代码如下：

```
1  Resources resources = this.getResources();
2  InputStream inputStream = null;
3  try {
4          inputStream = resources.openRawResource(R.raw.raw_file);
5          byte[] reader = new byte[inputStream.available()];
6          while (inputStream.read(reader) != -1) {
```

8.7 文件存储

8.2.3 资源文件

```
7           }
8           displayView.setText(new String(reader,"utf-8"));
9       } catch (IOException e) {
10          Log.e("ResourceFileDemo", e.getMessage(), e);
11      } finally {
12          if (inputStream != null) {
13          try {
14          inputStream.close();
15          }
16          catch (IOException e) { }
17          }
18      }
```

8.2 文件存储

8.2.3 资源文件

- 代码第8行的new String(reader,"utf-8")，表示以UTF-8的编码方式从字节数组中实例化一个字符串

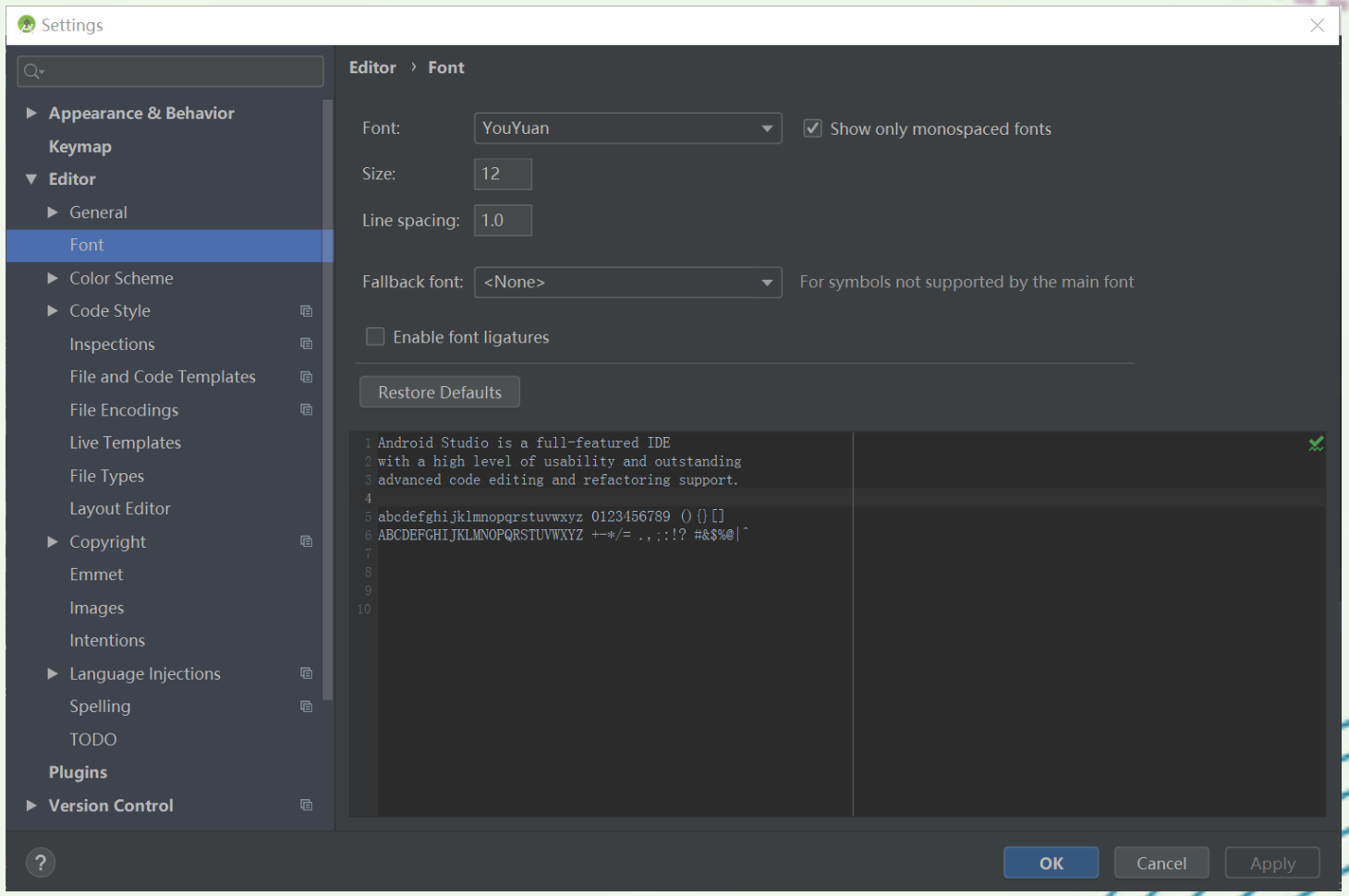
✓ 如果程序开发人员需要新建/res/raw/raw_file.txt文件，则需要选择使用UTF-8编码方式，否则程序运行时会产生乱码

✓ 选择的方法是在raw_file.txt文件上点击右键，选择“Properties”打开raw_file.txt文件的属性设置框，然后在“Resource”栏下的“Text file encoding”中，选择“Other：UTF-8”，如下图所示:

8.2 文件存储

8.2.3 资源文件

- 选择raw_file.txt文件编码方式



8.2 文件存储

8.2.3 资源文件

- /res/xml目录下的XML文件与其它资源文件有所不同，程序开发人员不能够以流的方式直接读取，其主要原因在于Android系统为了提高读取效率，减少占用的存储空间，将XML文件转换为一种高效的二进制格式

8.2 文件存储

8.2.3 资源文件

- 如何在程序运行时读取/res/xml目录下的XML文件
  - 首先在/res/xml目录下创建一个名为people.xml的文件
  - XML文件定义了多个<person>元素，每个<person>元素都包含三个属性name、age和height，分别表示姓名、年龄和身高
- /res/xml/people.xml文件代码如下:

```
1  <people>
2      <person name="李某某" age="21" height="1.81" />
3      <person name="王某某" age="25" height="1.76" />
4      <person name="张某某" age="20" height="1.69" />
5  </people>
```

8.7 文件存储

8.2.3 资源文件

- 读取XML格式文件
  - 首先通过调用资源实例的getXml()函数，获取到XML解析器XmlPullParser
  - XmlPullParser是Android平台标准的XML解析器，这项技术来自一个开源的XML解析API项目XMLPULL
- ResourceFileDemo示例中关于读取XML文件的核心代码如下:

```
1   XmlPullParser parser = resources.getXml(R.xml.people);
2   String msg = "";
3   try {
4           while (parser.next() != XmlPullParser.END_DOCUMENT) {
5       String people = parser.getName();
6       String name = null;
```

8.7 文件存储

8.2.3 资源文件

- ✓读取XML格式文件

```
7           String age = null;
8           String height = null;
9           if ((people != null) && people.equals("person")) {
10                  int count = parser.getAttributeCount();
11                  for (int i = 0; i < count; i++) {
12      String attrName = parser.getAttributeName(i);
13      String attrValue = parser.getAttributeValue(i);
14              if ((attrName != null) && attrName.equals("name")) {
15      name = attrValue;
16              } else if ((attrName != null) && attrName.equals("age")) {
17      age = attrValue;
18              } else if ((attrName != null) && attrName.equals("height")) {
19      height = attrValue;
20      }
21          }
```

8.7 文件存储

8.2.3 资源文件

- ✓读取XML格式文件

```
22            if ((name != null) && (age != null) && (height != null)) {
23            msg += "姓名："+name+"，年龄："+age+"，身高："+height+"\n";
24             }
25           }
26       }
27   } catch (Exception e) {
28          Log.e("ResourceFileDemo", e.getMessage(), e);
29   }
30   displayView.setText(msg);
```

8.7 文件存储

8.2.3 资源文件

- 读取XML格式文件
  - 代码第1行通过资源实例的getXml()函数获取到XML解析器
  - 第4行的parser.next()方法可以获取到高等级的解析事件，并通过对比确定事件类型，XML事件类型参考下表

事件类型	说明
START_TAG	读取到标签开始标志
TEXT	读取文本内容
END_TAG	读取到标签结束标志
END_DOCUMENT	文档末尾

8.7 文件存储

8.2.3 资源文件

- 读取XML格式文件
  - 第5行使用getName()函数获得元素的名称
  - 第10行使用getAttributeCount()函数获取元素的属性数量
  - 第12行通过getAttributeName()函数得到属性名称
  - 最后在第14行到第19行代码中，通过分析属性名获取到正确的属性值，并在第23行将属性值整理成需要显示的信息